Rozdział 1

Język SQL — część 1

SQL jest to język wysokiego poziomu do komunikacji z bazami danych (ściślej: z systemami zarządzania bazami danych). Język ten zorientowany jest na operowanie zbiorami i jak gdyby opisywanie wyniku. Podajemy „co ma być zrobione", a nie „jak to zrobić". To DBMS sam określa „najlepszy" sposób wykonania polecenia. Dzięki temu możliwa staje się dość wyrafinowana *optymalizacja wykonywania zapytań*.

1.1 Przykładowa baza danych

W naszych przykładach używać będziemy „zoologicznej" bazy danych o następującym schemacie

Gatunki(nazwa,kontynent,groźny,chroniony)
Zwierzaki(imię,gatunek,wiek,waga)
Potrawa(nazwa,koszt)
Jada(gatunek,potrawa,ile)

Podkreślone atrybuty oznaczają klucze główne.

1.2 Zapytania

Do zadawania zapytań służy tylko jedno polecenie: `SELECT`. Pełna jego składnia jest złożona, dlatego obejrzymy na razie uproszczoną postać

```
SELECT jakie atrybuty
FROM z jakich tabel
WHERE jakie warunki muszą spełniać wybrane
  wiersze
```

Zaczniemy od prostego zapytania: „Jak nazywają się lwy?" (czyli jakie są ich imiona).

```
SELECT imie
FROM Zwierzaki
```

```
WHERE gatunek = 'lew';
```

Wynikiem tego zapytania będzie

imie
Kocio
Puszek
...

Najprostszą realizację tego zapytania można opisać następująco:

1. Weź tabelę podaną we frazie FROM.
2. Wybierz wiersze używając warunku z frazy WHERE (selekcja).
3. Wybierz tylko kolumny wskazane frazą SELECT (rzutowanie).

Pierwsze bardziej formalne podejście do semantyki operacyjnej mogłoby wyglądać tak

1. Wprowadzamy *zmienną krotkową* (np. nazywającą się tak, jak tabela), przebiegającą po kolejnych wierszach (krotkach) tabeli.
2. Sprawdzamy czy „bieżący" wiersz spełnia warunek z frazy WHERE.
3. Jeśli tak, obliczamy wyrażenia we frazie SELECT używając tego wiersza i dopisujemy nowy wiersz do wyniku.

1.2.1 Fraza SELECT

Fraza SELECT musi być pierwszą frazą zapytania. Oprócz nazw kolumn i wyrażeń nad nimi można w niej używać dodatkowo specjalnego symbolu `*` oznaczającego „wszystkie atrybuty relacji". Na przykład aby wynik zawierał wszystkie kolumny tabeli piszemy:

```
SELECT *
FROM Zwierzaki
WHERE gatunek = 'lew';
```

i otrzymujemy

imie	**gatunek**	**wiek**	**waga**
Kocio	lew	4	120
Puszek	lew	7	87
...	...	...	...

1.2.2 Fraza WHERE

Fraza WHERE jest opcjonalna, jej pominięcie oznacza, że wynik będzie zawierał odpowiedniki wszystkich wierszy źródłowej tabeli.

W warunkach umieszczanych we frazie WHERE można umieszczać typowe wyrażenia złożone zawierające

- Operatory arytmetyczne: +, -, *, /.
- Operatory porównywania: =, <>, <, >, <=, >=.
- Spójniki logiczne: AND, OR, NOT.

na przykład:

```
SELECT imie
FROM Zwierzaki
WHERE gatunek = 'lew' AND wiek > 4;
```

1.2.3 Wartości puste: NULL

W bazie danych często przechowujemy niekompletną informację. Czasem jest to zamierzone, na przykład w tabeli z ocenami studentów z różnych przedmiotów pewne kolumny mogą nie zawierać ocen, ponieważ nie odbyły się jeszcze egzaminy.

W innych przypadkach informacja nie jest znana, na przykład może nie być wiadomo, kto jest producentem komputera otrzymanego w ramach darowizny, choć niewątpliwie taki producent istnieje.

Może się też zdarzyć tak, że dla pewnych wierszy w ogóle nie jest określona (nie dotyczy tych „obiektów"), np. zawartość kolumny „nazwisko panieńskie" w tabeli Studenci nie będzie określona dla mężczyzn.

We wszystkich takich sytuacjach używa się w SQL wyróżnionej wartości `NULL`, umieszczając ją w odpowiednich kolumnach w danym wierszu.

Należy bardzo uważać na wartości NULL w warunkach. Logika dla warunków w SQL jest *trójwartościowa*: **true**, **false**, **unknown**. Jakiekolwiek normalne porównanie z wartością NULL daje wynik **unknown**, podobnie jest dla operacji arytmetycznych (inaczej mówiąc NULL jest „zaraźliwe").

Dlatego do sprawdzania wartości pustych należy używać specjalnych operatorów porównania IS NULL i IS NOT NULL.

1.2.4 Inne warunki

W podanych dalej warunkach elementarnych SQL operator można zwykle poprzedzać dodatkowo symbolem `NOT` (oczywiście z odwróceniem znaczenia).

Wyrażenie

wartość `IN` *zbiór*

bada przynależność *wartości* do *zbioru*. Zbiór może byc podany jawnie przez wyliczenie elementów

($elem_1$, ..., $elemx_n$)

lub jako zapytanie wewnętrzne.

Wyrażenie

wartość `BETWEEN` *a* `AND` *b*

sprawdza, czy *wartość* należy do podanego przedziału domkniętego [a,b].

Wyrażenie

napis `LIKE` *wzorzec*

oznacza dopasowanie napisu do wzorca. We wzorcu (który także powinien być napisem) `%` oznacza dowolny ciąg znaków, zaś _ dowolny pojedynczy znak.

Wyrażenie

napis SIMILAR TO *wzorzec*

to rozszerzone dopasowanie do wzorców, używające we wzorcach wyrażeń regularnych.

1.2.5 Inne wyrażenia

W klauzuli `SELECT` można używać wyrażenia

```
CASE WHEN warunek THEN wartość
     ...
     ELSE wartość
END
```

na przykład do kategoryzacji wartości

```
SELECT imie, nazwisko,
       CASE WHEN wiek IS NULL THEN 'nie wiadomo'
            WHEN wiek >= 18 THEN 'dorosły'
            ELSE 'nieletni'
       END
FROM ...
```

Do wartości NULL przyda się wyrażenie

`COALESCE(`*v1*`,` *v2*`)`.

Jego wartością jest *v1*, o ile nie jest `NULL`, w przeciwnym razie *v2*.

1.2.6 Eliminacja powtórzeń

SQL nie jest algebrą relacji, dlatego powtórzenia nie są automatycznie eliminowane z tabel. Do usuwania potwórzeń z wyników zapytań służy modyfikator `DISTINCT` we frazie SELECT

```
SELECT DISTINCT kontynent
FROM Gatunki;
```

Przy braku DISTINCT każdy kontynent zostałby wypisany wielokrotnie.

Operacje teoriomnogościowe `UNION`, `INTERSECT` i `EXCEPT` automatycznie eliminują powtórzenia, o ile nie zastosowano modyfikatora `ALL`

```
(SELECT gatunek
 FROM Zwierzaki
 WHERE waga > 100)
UNION ALL
(SELECT gatunek
 FROM Zwierzaki
 WHERE wiek > 10);
```

1.3 Tworzenie tabel

Do tworzenia tabel służy konstrukcja CREATE TABLE

```
CREATE TABLE nazwa(
  kolumna typ więzy-spójności,
  ...
);
```

Deklarujemy w niej kolumny tabeli, ich typy oraz dodatkowe ograniczenia poprawności.

Tabele usuwamy konstrukcją DROP TABLE `DROP TABLE nazwa;`

Generalnie w SQL wszelkie polecenia tworzenia obiektów w bazie danych mają postać

CREATE *typ-obiektu nazwa ...*;

zaś polecenia usuwania

DROP *typ-obiektu nazwa*;

Utworzymy teraz niektóre tabele, których używaliśmy w przykładach zapytań. Zaczniemy od tabeli Gatunki

```
CREATE TABLE Gatunki (
  nazwa VARCHAR(30) PRIMARY KEY,
  kontynent VARCHAR(25),
  grozny BOOLEAN,
  chroniony BOOLEAN
);
```

Tworzenie tabeli Zwierzaki wygląda podobnie

```
CREATE TABLE Zwierzaki (
  imie VARCHAR(20) PRIMARY KEY,
  gatunek VARCHAR(30) REFERENCES Gatunki,
  wiek INTEGER,
  waga NUMERIC
);
```

1.3.1 Typy danych

Większość typów w SQL to typy znane z innych języków programowania. Najbardziej przydatne to

- `CHAR(`*n*`)`: napis o długości n znaków,
- `VARCHAR(`*n*`)`, `VARCHAR2(`*n*`)`: napis o zmiennej długości nie przekraczającej n znaków,
- `NUMERIC(`*n*`)`, `NUMERIC(`*n*`, `*m*`)`: liczba o zadanej precyzji
- `INTEGER`, `INT`: liczba całkowita,
- `DATE`: data,
- `BOOLEAN`: wartość logiczna (prawda lub fałsz).

SQL zawiera wiele funkcji do konwersji między typami, ale oprócz nich można używać uniwersalnej konstrukcji `CAST`, np.

```
CAST(wczoraj AS TEXT)
```

zamieni wartość typu `DATE` z kolumny `wczoraj` na tekst.

1.3.2 Tabele robocze

Tabele zdefiniowane przez CREATE TABLE są trwale przechowywane w bazie danych i aby się ich pozbyć należy użyć DROP TABLE. Czasem jednak potrzebujemy tabeli tylko na czas obliczeń do przechowywania wyników częściowych. Służa do tego tabele robocze.

Są one widoczne *tylko* w sesji, w której zostały utworzone i znikają automatycznie po jej zakończeniu. Tworzymy je używając rozszerzonej postaci CREATE TABLE

```
CREATE TEMPORARY TABLE nazwa(
  ...
);
```

albo po prostu zapisując wynik zapytania

```
SELECT ... INTO TEMPORARY TABLE nazwa
FROM ...
...;
```

1.3.3 Więzy spójności

Terminem *więzy spójności* określa się elementarne warunki na poprawność bazy danych, zapisane składniowo w definicji tabeli lub innego obiektu. Podaje się je po nazwie i typie kolumny której dotyczą. Jeśli natomiast ograniczenie dotyczy kilku kolumn (np. klucz główny składający się z dwóch kolumn), zapisujemy je osobną deklaracją.

Deklaracja `NOT NULL` zakazuje umieszczania wartości pustych w kolumnie, której dotyczy. Przy wstawianiu wierszy zawsze należy podać jakąś wartość dla tej kolumny.

Deklaracja `UNIQUE` mówi, że wartości w tej kolumnie (lub kolumnach) są unikalne — żadne dwa wiersze nie mogą zawierać tej samej wartości.

Deklaracja `PRIMARY KEY` określa klucz główny. Równoważna jest deklaracji `UNIQUE NOT NULL`, no i może wystąpić tylko raz.

Deklaracja `CHECK` *`warunek`* umożliwia podanie wyrażenia SQL, określającego dodatkowe ograniczenia na poprawność danych.

I wreszcie deklaracja `REFERENCES` *`nazwa-tabeli`*. Określa ona kolumnę jako *klucz obcy* (ang. *foreign key)*. Klucz obcy zawiera odwołanie do innej tabeli przez umieszczenie w deklarowanej kolumnie wartości z klucza głównego do tamtej tabeli. Oznacza to, że dozwolone są tylko takie wartości, które występują jako klucze w tabeli, do której się odwołujemy. Takie ograniczenie jest czasem nazywane „więzami integralności referencyjnej".

Oczywiście dla klucz obcego trzeba zadbać o to, żeby typy kolumn, po których łączymy, były zgodne.

W deklaracji tabeli Zwierzaki wystąpiła deklaracja kolumny gatunek

```
...
  gatunek VARCHAR(30) REFERENCES Gatunki,
...
```

Oznacza to, że kolumna `gatunek` może zawierać tylko wartości z klucza głównego tabeli `Gatunki`, czyli z kolumny `nazwa` w tej tabeli.

W deklaracjach odwołań do innych tabel można dodatkowo określić wymagane zachowanie w przypadku usuwania lub modyfikacji wartości klucza obcego w jego macierzystej tabeli, np.:

```
... ON DELETE SET NULL,
... ON UPDATE CASCADE
```

Pierwsza z tych deklaracji mówi, że w przypadku usunięcia danej wartości klucza obcego z macierzystej tabeli należy jego wartość zastąpic przez NULL. Przy braku takiej deklaracji usunięcie tej wartości spowodowałoby błąd.

Druga deklaracja mówi, że w przypadku zmiany wartości klucza na inną należy zmienioną wartość umieścić również w miejscach odwołań (tzw. „kaskada modyfikacji". Wartość klucza jest zresztą zmieniana niezwykle rzadko, na ogół jako wynik błędu przy wpisywaniu.

1.3.4 Cykliczne zależności referencyjne

W przypadku kluczy obcych należy zwracać uwagę na ewentualne cykle. Powstający problem jest odmianą powszechnie znanego problemu „kury i jajka" (co było pierwsze?). Poniższe dwa polecenia zawsze powodują błąd

```
CREATE TABLE Kura (
  imie CHAR(8) PRIMARY KEY,
  jajko INTEGER REFERENCES Jajko
);
CREATE TABLE Jajko (
  numer INTEGER PRIMARY KEY,
  kura CHAR(8) REFERENCES Kura
);
```

W przypadku deklaracji klucza obcego wymaga się, aby istniała tabela, do której się odwołujemy. Tutaj podczas tworzenia tabeli `Kura` system napotyka frazę odnoszącą się do tabeli `Jajko`, która jeszcze nie istnieje!

Zmiana kolejności poleceń też nic nie pomoże, ponieważ analogiczne odwołanie występuje w tabeli Jajko.

1.4 Polecenia modyfikacji schematu

Aby poradzić sobie z tym problemem musimy sięgnąć do poleceń modyfikacji schematu bazy danych. Utworzymy najpierw tabele bez określania więzów kluczy obcych:

```
CREATE TABLE Kura (
  imie CHAR(8) PRIMARY KEY,
  jajko INTEGER
);
CREATE TABLE Jajko (
  numer INTEGER PRIMARY KEY,
  kura CHAR(8)
);
```

Nowe więzy do istniejącej tabeli można dodać poleceniem:

```
ALTER TABLE tabela
  ADD CONSTRAINT nazwa ograniczenie;
```

W naszym przypadku potrzebne będą dwa polecenia:

```
ALTER TABLE Kura ADD CONSTRAINT Kura_Jajko
  FOREIGN KEY (jajko) REFERENCES Jajko(numer)
  INITIALLY DEFERRED DEFERRABLE;
ALTER TABLE Jajko ADD CONSTRAINT Jajko_Kura
  FOREIGN KEY (kura) REFERENCES Kura(imie)
  INITIALLY DEFERRED DEFERRABLE;
```

Fraza `INITIALLY DEFERRED DEFERRABLE` żąda od SQL *odroczenia sprawdzania więzów* do chwili zatwierdzenia transakcji, np. aby wstawić ('Czubatka', 1) do tabeli `Kura` i (1, 'Czubatka') do tabeli `Jajko` użyjemy:

```
INSERT INTO Kura VALUES ('Czubatka', 1);
INSERT INTO Jajko VALUES (1, 'Czubatka');
COMMIT;
```

Bez opóźnionego sprawdzania więzów nie można byłoby wstawić żadnego wiersza do tabel `Kura` ani `Jajko`, ponieważ już pierwszy `INSERT` naruszałby więzy, chyba że dopuścimy wartości puste (`NULL`) w kolumnie klucza obcego.

1.4.1 Usuwanie więzów

Nazwane więzy można usuwać poleceniem:

```
ALTER TABLE tabela DROP CONSTRAINT nazwa;
```

Należy zawsze pamiętać, aby przed usunięciem tabel zawsze przedtem usunąć ręcznie więzy cykliczne, w przeciwnym razie SQL nie pozwoli na usunięcie tabel.

```
ALTER TABLE Jajko DROP CONSTRAINT Jajko_Kura;
ALTER TABLE Kura DROP CONSTRAINT Kura_Jajko;
DROP TABLE Jajko;
DROP TABLE Kura;
```

W DROP TABLE można użyć modyfikatora CASCADE, ale nie zawsze radzi on sobie z takimi sytuacjami.

1.5 Funkcje agregujące

Funkcje agregujące są przeznaczone do obliczania wartości parametrów „statystycznych", takich jak średnia czy suma, dotyczących całej tabeli (lub wybranych grup wierszy), a nie pojedynczych wierszy.

```
SELECT AVG(waga)
FROM Zwierzaki
WHERE gatunek = 'Niedźwiedź';
```

oblicza średnią wagę niedźwiedzi, czyli średnią z wartości w kolumnie `waga` dla wierszy zawierających `'Niedźwiedź'` w kolumnie `gatunek`.

Standardowe funkcje agregujące to `AVG`, `COUNT`, `MAX`, `MIN` i `SUM`. Z wyjątkiem wyrażenia `COUNT(*)` wartości puste są pomijane.

Funkcji `COUNT` warto przyjrzeć się dokładniej. Zlicza ona wiersze i często ma argument zastępczy `*`:

```
SELECT COUNT(*)
FROM Zwierzaki
WHERE gatunek = 'Niedźwiedź';
```

Jeśli zamiast `*` jej argumentem jest nazwa kolumny, to nie są liczone wiersze, zawierające tej kolumnie wartości puste.

Natomiast poprzedzenie takiego argumentu dodatkowo modyfikatorem `DISTINCT` spowoduje obliczenie, ile *różnych* wartości występuje w tej kolumnie, na przykład

```
SELECT COUNT(DISTINCT gatunek)
FROM Zwierzaki;
```

policzy, ile mamy różnych gatunków w tabeli Zwierzaki.

1.6 Grupowanie

Dzielenie wierszy na grupy frazą `GROUP BY` ułatwia równoczesne obliczanie parametrów statystycznych dla wybranych podzbiorów wierszy. Zapytanie

```
SELECT gatunek, AVG(waga)
FROM Zwierzaki
GROUP BY gatunek;
```

poda średnią wagę dla każdego gatunku w tabeli Zwierzaki.

Zauważmy, że eliminację powtórzeń można rzeprowadzić grupowaniem zamiast używać `DISTINCT`:

```
SELECT kontynent
FROM Gatunki
GROUP BY kontynent;
```

Warunkiem frazy `WHERE` można ograniczyć grupowanie tylko do wybranych wierszy

```
SELECT gatunek, AVG(waga)
FROM Zwierzaki, Gatunki
WHERE Zwierzaki.gatunek = Gatunki.nazwa
  AND kontynent = 'Afryka'
GROUP BY gatunek;
```

Czasem jednak chcemy dla całych grup, a nie pojedynczych wierszy. Służy do tego fraza `HAVING`. Polecenie

```
SELECT gatunek, AVG(waga)
FROM Zwierzaki, Gatunki
WHERE Zwierzaki.gatunek = Gatunki.nazwa
GROUP BY gatunek
HAVING COUNT(*) > 2;
```

odrzuci wszystkie grupy zawierające mniej niż 3 elementy.

Uwaga: Chcąc znaleźć najwyższą średnią po grupach, nie możemy po prostu napisać `MAX(AVG(`*`wyrażenie`*`))` [Oracle akceptuje taką konstrukcję, ale nie jest to zgodne ze standardem SQL]. Można jednak napisać proste zapytanie zagnieżdżone:

```
SELECT MAX(średnia_z_ocen)
FROM (SELECT AVG(ocena) AS średnia_z_ocen
      FROM Oceny
      GROUP BY indeks) Średnie;
```

1.7 Zadania

Baza danych biblioteki jest oparta na następującym schemacie:

```
CREATE TABLE Ksiazki (
  nrk NUMERIC(5) PRIMARY KEY,
  tytul VARCHAR(20) NOT NULL,
  autor VARCHAR(25),
  wydawca VARCHAR(20),
  rok_wyd NUMERIC(4),
  data_zakupu DATE,
  cena NUMERIC(6,2));

CREATE TABLE Czytelnicy (
  nrcz NUMERIC(4) PRIMARY KEY,
  nazwisko VARCHAR(20) NOT NULL,
  imie VARCHAR(15) NOT NULL,
  zawod VARCHAR(15));

CREATE TABLE Wypozyczenia (
  nrk NUMERIC(5) NOT NULL REFERENCES Ksiazki,
  nrcz NUMERIC(4) NOT NULL REFERENCES Czytelnicy,
  data_wyp DATE NOT NULL,
  data_zwr DATE,
  PRIMARY KEY(nrk, nrcz, data_wyp));
```

Zapisz w SQL następujące zapytania:

Ćwiczenie 1.1. Która obecnie wypożyczona książka jest najdłużej trzymana i przez kogo (może być kilka takich książek — należy podać wszystkie)? Podaj autora, tytuł oraz imię i nazwisko czytelnika.

Rozwiązanie.

```
SELECT autor,tytul,imie,nazwisko
FROM Wypozyczenia JOIN Ksiazki ON Wypozyczenia.nrk = Ksiazki.nrk
     JOIN Czytelnicy ON Wypozyczenia.nrcz = Czytelnicy.nrcz
```

```
WHERE data_zwr IS NULL
  AND data_wyp = (SELECT MIN(data_wyp)
                  FROM Wypozyczenia
                  WHERE data_zwr IS NULL);
```

albo optymistycznie ze złączeniem naturalnym

```
SELECT autor,tytul,imie,nazwisko
FROM Wypozyczenia NATURAL JOIN Ksiazki NATURAL JOIN Czytelnicy
WHERE data_zwr IS NULL
  AND data_wyp = (SELECT MIN(data_wyp)
                  FROM Wypozyczenia
                  WHERE data_zwr IS NULL);
```

Ćwiczenie 1.2. Kto czytał najdroższą książkę wydaną przed 1989 rokiem (może być kilka takich książek — podaj dla wszystkich imię i nazwisko czytelnika)?

Rozwiązanie. Warto użyć perspektywy (być może lokalnej, ale o tym później)

```
CREATE VIEW Stare_ksiazki AS
SELECT * FROM Ksiazki
WHERE rok_wyd < 1989;

SELECT imie,nazwisko
FROM Wypozyczenia NATURAL JOIN Stare_ksiazki NATURAL JOIN Czytelnicy
WHERE cena = (SELECT MAX(cena)
                     FROM Stare_ksiazki);
```

Ćwiczenie 1.3. Podaj numery katalogowe i tytuły pięciu (lub więcej, jeśli jest „remis") książek o największej liczbie wypożyczeń.

Rozwiązanie.

```
CREATE VIEW Ile_Wypozyczen AS
SELECT nrk,COUNT(*) AS ile
FROM Wypozyczenia
GROUP BY nrk;

SELECT tytul,Ksiazki.nrk
FROM Ile_Wypozyczen JOIN Ksiazki ON Ile_Wypozyczen.nrk = Ksiazki.nrk
WHERE ile in (SELECT ile
              FROM Ile_Wypozyczen
              ORDER BY ile DESC LIMIT 5);
```

W bazie danych znajdują się tabele:

```
CREATE TABLE Osoby (
  id NUMERIC(5) PRIMARY KEY,
  nazwisko VARCHAR(20) NOT NULL,
  imie VARCHAR(15) NOT NULL,
  miasto VARCHAR(20));

CREATE TABLE Agenci (
  id NUMERIC(4) PRIMARY KEY,
  imie VARCHAR(15) NOT NULL,
  nazwisko VARCHAR(20) NOT NULL);

CREATE TABLE Ubezpieczenia (
  polisa NUMERIC(5) PRIMARY KEY,
  data_od DATE NOT NULL,
  data_do DATE NOT NULL CHECK (data_do > data_od),
  wariant CHAR(1),
  ag_id NUMERIC(4) NOT NULL REFERENCES(Agenci),
  os_id NUMERIC(5) NOT NULL REFERENCES(Osoby));
```

Zapisz w SQL następujące polecenia:

Ćwiczenie 1.4. Jaka jest maksymalna liczba ubezpieczeń jednej osoby?

Rozwiązanie.

```
SELECT MAX(ile)
FROM (SELECT COUNT(*) AS ile
      FROM Ubezpieczenia
      GROUP BY os_id) maksy;
```

Ćwiczenie 1.5. Który agent nie zawarł żadnego ubezpieczenia? Podaj jego imię i nazwisko (może być kilku takich agentów).

Rozwiązanie.

```
SELECT imie,nazwisko
FROM Agenci LEFT JOIN Ubezpieczenia ON id=ag_id
GROUP BY id,imie,nazwisko
HAVING COUNT(polisa)=0;
```

Ćwiczenie 1.6. Który klient ma najwięcej ubezpieczeń? Podaj imię, nazwisko i liczbę ubezpieczeń. Uwaga: może być kilku o tej samej liczbie — wtedy należy podać wszystkich!

Rozwiązanie.